IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application No.	:	09/431,449	Confirmation No.	:	8736
Applicant	:	GALLUSCIO et al.			
Filed	:	November 1, 1999			
TC/A.U.	:	2194			
Examiner	:	HOANG, Phuong N.			
Docket No.	:	6572-14			
Customer No.	:	39207			

APPEAL BRIEF UNDER 37 C.F.R. § 41.37

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REAL PARTY IN INTEREST

The patent application to which the present appeal brief pertains, namely patent application no. 09/431,449, is assigned to Harris-Exigent, Inc. of Melbourne, Florida. A further real party in interest is Harris Corporation, of Melbourne, Florida.

RELATED APPEALS AND INTERFERENCES

There are no related interferences, appeals or judicial proceedings known to Appellants, Appellants' legal representative, or the assignee which are related to this matter.

Certificate Under 37 C.F.R. 1.8(a)

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as First Class Mail in an envelope addressed to: Mail Stop Appeal Brief-Patents, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on 6-19-06 Typed name of person signing this certificate: ROBERT J. SACCO

STATUS OF CLAIMS

Claims 1-20 are pending in the application and stand rejected. Claims 1-20 are appealed.

STATUS OF AMENDMENTS

An amendment filed prior to the final Office action was entered. The claims presented herein are in the form in which they were presented to the Examiner in the amendment.

SUMMARY OF THE CLAIMED SUBJECT MATTER

The present invention relates to a method and apparatus for high speed interprocess communications ("IPC"). In conventional IPC, multiple processes can communicate with one another via a shared region of random access memory ("RAM") to which each process can write data, and from which each process can read data. When communicating through the shared region of RAM, a first process functioning as the message source can write a message to the shared region of RAM. Subsequently, the second process, functioning as a message receiver, can read the written message from the shared region of RAM. Thus, at a minimum, two system calls are required to move n bytes of data from the first process to the second process through the shared region of RAM. Moreover, 2*n bytes of data will be stored in total – n bytes into the shared region of RAM, and n bytes into user memory space associated with the second process.

To overcome the excessive overhead associated with conventional IPC utilizing a shared region of RAM, the high speed IPC method and apparatus of the present invention avoids moving 2*n bytes of data by passing to the second process a memory offset which can be used by the second process to access the data. Importantly, the second process can manipulate and modify the data in place within the shared region of RAM. Accordingly, it is not required that the data be copied from the shared region of RAM to an alternate location while the manipulation takes place, thereby improving system efficiency.

Claim 1 recites a method for high speed interprocess communications comprising the step of attaching first and second processes (11, 12) to a message buffer (25) in a shared region (15) of random access memory (RAM) exclusive of operating system kernel space (14), each said process having a message list. (p. 8, lines 15-28; p. 9, line 29 – p. 10, line 2; p. 10 lines 24-27). In addition, the claimed method recites accumulating message data (30) from the first process in a location in the message buffer (25), and adding to the message list of said second process a memory offset (29) corresponding to the location in the message buffer. (p. 8, lines 26-28; p. 10, lines 10-15). Claim 1 further recites manipulating in the second process the accumulated data at said location corresponding to said offset. (p. 12, lines 19-21). The accumulated message data is transferred from the first process to the second process with minimal data transfer overhead. (p. 6, lines 1-3; p. 15, lines 11-13).

Claims 2 and 3 depend from claim 1. Claim 2 recites that the attaching step comprises the following steps: detecting a previously created shared region of RAM; if a shared region of RAM is not detected, creating and configuring a shared region of RAM for storing accumulated data; and, attaching the first and second processes to the shared region. (p. 4, lines 6-10; p. 13, lines 1-21). Claim 3 recites that the message list is a message queue. (p. 8, lines 25-29).

Claim 4 depends from claim 3, and recites that the adding step comprises the steps of retrieving a memory offset in the message buffer corresponding to the location of data accumulated by the first process, and inserting the memory offset in the message queue corresponding to the second process. (p. 13, line 28 - p. 14, line 1; p. 14 lines 13-17). Claim 5 further recites that the inserting step comprises the step of atomically assigning the memory offset to an integer location in the message queue corresponding to the second process. (p. 14, lines 18-19).

Claim 6 depends from claim 1, and recites that the manipulating step comprises the following steps: identifying a memory offset in the message list corresponding to the second process; processing in the second process message data stored at a location in the message buffer corresponding to the memory offset; and releasing the message buffer. (p. 4, line 24 – p. 5, line 1; p. 14, line 22 - p. 15, line 3). Claim 19 depends from claim 1 and recites the step of locking the accumulated data to prevent the first process

from accessing the accumulated data while the accumulated data is being manipulated. (p. 13, lines 2-14).

Claim 7 recites a method for configuring high speed interprocess communications between first and second processes (11, 12). The method includes the steps of disposing a message buffer (25) in a shared region (15) of random access memory (RAM) shared between the first and second processes and accumulating message data (30) from said first process in a location in the message buffer. (p. 9, lines 2-4; p. 13, line 9; p. 10, lines 10-11). Claim 7 also recites adding to a message list corresponding to the second process a memory offset corresponding to the location in the message buffer. (p. 10, lines 11-15). In addition, claim 7 recites manipulating in the second process the accumulated message data stored in the message buffer at a location corresponding to the offset. (p. 12, lines 19-21). The accumulated message data is transferred from said first process to said second process with minimal data transfer overhead. (p. 6, lines 1-3; p. 15, lines 11-13).

Claim 8 depends from claim 7, and further recites that the disposing step comprises creating and configuring a message buffer in a shared region (15) of RAM exclusive of operating system kernel space (14). (p. 8, lines 1-4; p. 13, lines 9-10). Claim 8 also recites creating a message list in the shared region for each process whereby the message list can store memory offsets (29) of message data (30) stored in the message buffer. (p. 13, lines 20-23). Claim 9 recites that the message list is a message queue. (p. 8, lines 25-26).

Claim 10 recites that the adding step comprises retrieving a memory offset in the message buffer, the memory offset corresponding to the location of the message data accumulated by the first process. (p. 13, line 28 - p. 14, line 1). Claim 10 further recites inserting the memory offset in the message queue corresponding to the second process. (p. 14 lines 15-17). Claim 11 recites that the inserting step comprises the step of atomically assigning the memory offset to an integer location in the message queue corresponding to the second process. (p. 14, lines 18-19).

Claim 12 depends from claim 7 and recites the manipulating step comprises the following steps: identifying a memory offset in the message list corresponding to the second d process; processing in the second process the accumulated message data at

a location in the message buffer corresponding to the memory offset; and releasing said message buffer. (p. 4, line 24 – p. 5, line 1; p. 14, line 22 - p. 15, line 3). Claim 20 depends from claim 7 and recites the step of locking the accumulated data to prevent the first process from accessing the accumulated data while the accumulated data is being manipulated. (p. 13, lines 2-14).

Claims 13-18 recite a computer apparatus comprising means (e.g. RAM (15), CPU and program code) for performing the steps recited in claims 1-6. (*See*, *e.g.*, p. 9, lines 14-23; p. 13, lines 4-7; p. 14, lines 5-12).

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Claims 7 and 9-12 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Erickson, U.S. Patent No. 6,181,707 in view of Brookler, et al., U.S. Patent No. 6,754,666. Claims 1-6, 8 and 13-18 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Erickson, U.S. patent no. 6,181,707 in view of Brookler, et al., U.S. patent no. 6,754,666, and further in view of Carter, U.S. patent no. 6,148,377. Claims 19 and 20 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Erickson, U.S. patent no. 6,181,707 in view of Brookler, et al., U.S. patent no. 6,754,666 in view of Carter, U.S. patent no. 6,148,377, and further in view of Bohannon, U.S. patent no. 5,991,845.

ARGUMENT

The following claim groupings identify groups of claims which are different in scope and believed to be separately patentable. Accordingly, the claims of any particular group do not stand or fall with claims of any other group.

A. Claims 1 and 13

1. The cited references do not teach or suggest the step of a first process adds to a message list corresponding to the second process a memory offset which corresponds the location of data in the message buffer.

In rejecting independent claims 1 and 13, the Examiner states that Erickson, et al. teaches a method for high speed interprocess communications comprising the steps of: a) attaching first and second processes to a message buffer in a shared region of

random access memory (RAM), each the process having a message list (col. 7, lines 4-55); b) accumulating message data from the first process in a location in the message buffer (col. 7, lines 15-20); and c) adding to the message list of the second process a memory offset corresponding to the location of the message buffer (col. 7, lines 52-57).

However, with all due respect to the Examiner, Erickson discloses a process which is entirely different than the claimed invention. Erickson discloses multiple matrix cards (2) and configuration cards (4) communicating via a TDM Bus. Col. 4, lines 53-59. Each of the cards has its own microprocessor, CDCC and dual-port RAM. Col. 6, lines 35-53. Notably, Erickson has utilized the same identifier (20) to identify dual-port RAM contained on different matrix cards. Indeed, Erickson teaches that data is exchanged between the cards, col. 6, lines 45-53, but not that the cards share data at a single location as claimed by Appellants.

Moreover, Carter, et al. also does not teach or suggest a first process adding a memory offset to a message list of a second process as is recited in Applicants' claims 1 and 13. Carter, et al. fails to satisfy the express "memory offset" language of Appellants' claims. Instead, the data in Carter, et al. is passed by value and not by reference from process to process. For example, Carter, et al. teaches that the system can include a coherent replication controller for generating a copy, or select number of copies, of a portion of the addressable memory space maintained in the local persistent memory device of a first computer and for storing the copy in the local persistent memory device of a second computer. (Col. 3, lines 41-46). The Appellants explicitly state, "Processes are notified of the location of the message data rather than actually receiving a copy of the message data." (Appellants' specification, page 8, lines 18-19). Therefore, the Carter, et al. reference fails to teach the passage of data by reference between processes with the use of memory offsets.

In contrast, the second process of Appellants' claimed invention can access the shared data in RAM with the use of memory offsets. Independent claim 1 recites that the first process adds to the message list of the second process a memory offset corresponding to the location in the message buffer. Further still, independent apparatus claim 13 recites a "means for said first process to add to said message list of

said second process a memory offset corresponding to said location in said message buffer

2. <u>The cited references do not teach or suggest the step of manipulating data in the second process at the location corresponding to the memory offset.</u>

In furtherance of the rejection of claims 1 and 13, the Examiner states that Brookler, et al. teaches the step of data manipulation, including manipulating in a process data at the location corresponding to the offset, the manipulation modifying the data in place at the location (in place schema and data manipulation operations, col. 15, line 24 and col. 16, line 14). The Examiner reasons that it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Erickson, et al. and Brookler, et al. because the accumulated data/imported data can be normalized and cleansed (col. 15, lines 28-32). However, any in-place schema and data manipulation to the DBMS schema and data are occurring <u>exclusively</u> outside of RAM, namely within disk space.

3. <u>The cited references do not teach or suggest that the shared region of RAM is exclusive of the operating system kernel space</u>

In even greater furtherance of the rejection of claims 1 and 13, the Examiner states that "Carter, et al. teaches the step of RAM is exclusive to the operating system (fig. 1 and 2). " The Examiner's meaning here is not entirely clear. However, the Examiner appears to suggest that Carter et al. discloses the recitation in claims 1 and 13 that "the shared region of random access memory (RAM) is exclusive of operating system kernel space". The Examiner reasons that it would have been obvious to one of ordinary skill in the art to combine the teachings of Erickson, et al., Brookler, et al. and Carter, et al. because the RAM of Carter, et al. is exclusive to the operating system and would speed up the transferring and accessing data process when it does not have to access the operating system.

However, Carter, et al. also fails to make up for the deficiencies found in Erickson, et al. and Brookler, et al. regarding Applicants' independent claims 1 and 13. Carter, et al. discloses distributed shared memory systems and processes that can

connect into each node of a computer network. The system encapsulates the memory management operations of the connected nodes and provides an abstraction of a shared virtual memory that can span across each node of the network. Optionally, the shared memory can span across each memory device connected to a computer network. Accordingly, each node on the network having the distributed shared memory system of the invention can access the shared memory. "To establish *prima facie* obviousness of a claimed invention, all the claim limitations must be taught or suggested by the prior art." MPEP § 2143.03, *citing In re Royka*, 490 F.2d 981, 985 (CCPA 1970); *see also CFMT, Inc. v. YieldUp Int'l Corp.*, 349 F.3d 1333, 1342 (Fed. Cir. 2003). Consequently, the Examiner has failed to establish a *prima facie* case of obviousness and claims 1 and 13 are distinguishable from the cited art. Accordingly, the rejection of independent claims 1 and 13 is improper and must be reversed.

B. Claim 7

1. The cited references do not teach or suggest a first process adds to a message list corresponding to the second process a memory offset which corresponds the location of data in the message buffer.

In rejecting claim 7, the Examiner states that Erickson, et al. teaches a method for configuring high speed interprocess communication between first and second processes (embedded control applications running on the processors/cards) comprising the steps of: a) disposing a message buffer in a shared region of RAM shared between the first and second processes (col. 7, lines 4-55); b) accumulating message data from the first process in a location in the message buffer (col. 7, lines 15-20); c) adding to the message list of the second process a memory offset corresponding to the location in the message buffer (col. 7, lines 52-57); and d) whereby the accumulated message data is transferred from the first process to the second process the accumulated data at the location corresponding to the offset, the manipulation modifying the accumulated data in place at the location.

However, Erickson, et al. does not teach the first process adding to a message list corresponding to the second process a memory offset which corresponds to the location of data in the message buffer as the Examiner has stated. Erickson, et al. teaches the microprocessor of the source card loads an offset into its CDCC's Word-In-

Queue Register. This offset corresponds to the offset of the last message word that is stored in dual-port RAM. The CDCC's transmit hardware portion offsets the base address of its transmit queue in dual-port RAM with the Word-In-Queue Register and uses that combined address to fetch the first word of the message from its own transmit queue in dual-port RAM. As the message is sent, the Word-In-Queue is then regenerated at the receiver and the receiver uses this address offset to place the message data into the same-numbered location(s) in its receive queue. (Col. 7, lines 44-55). By loading its Word-In-Queue Register, the source microprocessor triggers the process whereby the CDCC transmit hardware portion automatically dispatches the message data, one word per TDM frame, onto the TDM bus. Such a process requires many more steps and is far more resource intensive than the claimed method of high speed interprocess communication. Put simply, the source card does not add to a message list corresponding to the receiver a memory offset corresponding to the location of the message in the message buffer.

Moreover, Erickson, et al. does not have a message buffer in a shared region of random access memory (RAM) as is also required by claim 7. Erickson, et al. discloses multiple matrix cards (2) and configuration cards (4) communicating via a TDM Bus. (Col. 4, lines 53-59). Each of the cards has its own microprocessor, CDCC and dual-port RAM. (Col. 6, lines 35-53). Notably, Erickson, et al. has utilized the same identifier (20) to identify dual-port RAM contained on different matrix cards. Indeed, Erickson, et al. teaches that data is exchanged between the cards, (col. 6, lines 45-53), but not that the cards share data at a single location as claimed by Appellants.

Further, Brookler, et al. also does not teach or suggest a first process adding a memory offset to a message list of a second process. In fact, Brookler, et al. makes <u>no</u> mention of a memory offset, message list, or equivalents thereof.

2. <u>The cited references do not teach or suggest manipulating data in the second process at the location corresponding to the memory offset.</u>

In furtherance of the rejection of claim 7, the Examiner states that Brookler, et al. teaches data manipulation, including manipulating in a process accumulated data (imported data) at the location corresponding to the offset, the manipulation modifying the accumulated data in place at the location (in place schema and data manipulation

operations, col. 15, line 24 - col. 16, line 14). The Examiner reasons that it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Erickson, et al. and Brookler, et al. because the accumulated data/imported data can be normalized and cleansed (col. 15, lines 28-32).

The Brookler, et al. invention describes a method of efficiently storing data items in a database management system ("DBMS"). The DBMS data is initially stored in the disk drive space (col. 7, lines 50-53). The DBMS includes data stored therein as well as schema and data stored in one or more tables defined in the schema (Id.). Computer-executable process <u>steps</u> of the DBMS are transferred from the disk space via the computer bus to RAM and executed therefrom by the CPU (col. 7, lines 45-49). It is important to note the difference between DBMS process steps and DBMS schema and data. The DBMS schema and data are <u>not</u> being transferred to RAM; only the DBMS process steps are transferred to RAM. Furthermore, any in-place schema and data manipulation to the DBMS schema and data are occurring <u>exclusively</u> outside of RAM, namely within disk space. Since Brookler, et al. does not access the message buffer in shared RAM to manipulate the data contained within the location corresponding to a memory offset, it is for this reason that Brookler, et al. does not mention or suggest memory offsets or their use.

The Brookler, et al. discloses two possible approaches for accessing DBMS data stored in the disk space. The first approach is an SQL approach in which SQL language is used to access "disk-based data" that is stored and managed by the DBMS. This continues to indicate that the database data in Brookler, et al. is not pointed to a location in RAM using a memory offset as is taught by Appellants' method claim 7. The second approach is a "memory-based" approach read from the DBMS and then stored in memory. However, Brookler, et al. admits that such a memory-based approach has the disadvantage that is requires "substantially more memory than the first approach." (Col. 16, lines 15-25). The second approach would indicate at the very least 2*n bytes of data would be stored, a problem that goes to the crux of Appellants' invention.

Because neither Erickson, et al. nor Brookler, et al. teach or suggest the recited step of a first process adds to a message list corresponding to the second process a memory offset which corresponds the location of data in the message buffer, or the

recited step of manipulating data with the second process at the location corresponding to a memory offset, the Examiner has not established a *prima facie* case of obviousness. Accordingly, the rejection of independent claim 7 is improper and must be reversed. *In re Ochiai*, 71 F.3d 1565, 1569 (Fed. Cir. 1995).

C. Claims 3, 9 and 15

1. The cited references do not teach or suggest the message list is a message queue.

In rejecting claims 3, 9 and 15, the Examiner states that Erickson teaches the message list is a message queue (col. 7, lines 12-55). However, with all due respect to the Examiner, Erickson does not have a message list. Erickson does have a receive queue 210 but it is not equivalent to the message list in the present invention wherein the message list is a message queue. Each message queue 23, 28 is a list of messages which can handle integer values of the message offset in a first-in-first out (FIFO) order. (Appellant's Specification, p. 10, lines 24-27 and p. 11, line1). In contrast, the receiver queue 210 in Erickson receives message data only (col. 7, lines 46-60), not an offset address of accumulated message data in a location in the small message buffer 25. Thus, the receiver queue 210 in Erickson is neither a message list nor a message queue and the claim limitations of claims 3, 9 and 15 are not met. Consequently, the Examiner has failed to state a *prima facie* case of obviousness and claims 3, 9 and 15 are distinguishable over the cited art. In addition, appellant believes claims 3, 9 and 15 are in a condition for allowance by virtue of their dependence on an allowable base claim. Accordingly, the rejection of claims 3, 9 and 15 must be reversed.

D. Claims 4, 10 and 16

1. The cited references do not teach or suggest the adding means comprises means for retrieving a memory offset in the message buffer corresponding to the location of the data accumulated by the first process and means for inserting the memory offset in the message queue corresponding to the second process.

In rejecting claims 4, 10 and 16, the Examiner states that Erickson teaches the adding means comprises means for retrieving a memory offset in the message buffer

corresponding to the location of the data accumulated by the first process and means for inserting the memory offset in the message queue corresponding to the second process. As previously discussed, the receiver queue 210 in Erickson receives message data only and does not receive a memory offset. The CDCC's transmit hardware portion offsets the base address of the transmit queue 205 with the Word-In-Queue Register 112 and uses that combined address to fetch the first word of the message from its own transmit queue 205 in dual port RAM 20. (See col. 7, lines 46-51). Thus, the claim limitations of claims 4, 10 and 16 are not met and the Examiner has failed to state a *prima facie* case of obviousness. In addition, appellant believes claims 4, 10 and 16 are in a condition for allowance by virtue of their dependence on an allowable base claim. Accordingly, the rejection of claims 4, 10 and 16 must be reversed.

E. Claims 5, 11 and 17

1. The cited references do not teach or suggest atomically assigning the memory offset to an integer location in the message queue corresponding to the second process.

In rejecting claims 5, 11 and 17, the Examiner states that Erickson teaches atomically assigning the memory offset to an integer location in the message queue corresponding to the second process. Appellant respectfully disagrees.

As discussed, the offset of the base address in Erickson, et al. of the transmit queue 205 with the Word-In-Queue Register 112 used to fetch the first word of the message from the transmit queue 205 is not assigned or placed into the receiver queue 210. Nor does this offset of the base address of the transmit queue 205 with the Word-In-Queue Register 112 correspond to a process like the second process of the present invention. In addition, applicant can find no mention of an integer location in the receiver queue 210 in Erickson. Thus, the claim limitations of claims 5, 11 and 17 are not met. Consequently, the Examiner has failed to state a *prima facie* case of obviousness and the claims are distinguishable over the cited references. In addition, appellant believes claims 5, 11 and 17 are in a condition for allowance by virtue of their dependence on an allowable base claim. Accordingly, the rejection of claims 5, 11 and 17 must be reversed.

F. Claims 6,12 and 18

1. The cited references do not teach or suggest a means for identifying a memory offset in the message list corresponding to the second process, means for using in the second process message data at a location in the message buffer corresponding to the memory offset, and means for releasing the message buffer.

In rejecting claims 6, 12 and 18, the Examiner states that Erickson teaches a means for identifying a memory offset in the message list corresponding to the second process, means for using in the second process message data at a location in the message buffer corresponding to the memory offset, and means for releasing the message buffer. Appellant respectfully disagrees.

As discussed, Erickson does not teach a memory offset in a message list. Erickson teaches that the offset of the base address of the transmit queue 205 with the Word-In-Queue Register 112 is used to fetch the first word of the message from the transmit queue 205 and is not assigned or placed into the receiver queue 210. Further, as discussed, Erickson does not have a message list. Consequently, Erickson cannot have a means for using message data located in a message buffer corresponding to the memory offset located in the message list. Thus, the claim limitations of claims 6, 12 and 18 are not met. Consequently, the Examiner has failed to state a *prima facie* case of obviousness and the claims are distinguishable over the cited references. In addition, appellant believes claims 6, 12 and 18 are in a condition for allowance by virtue of their dependence on an allowable base claim. Accordingly, the rejection of claims 6, 12 and 18 must be reversed.

G. Claim 8

1. The cited references do not teach or suggest creating a message list in the shared region for each process, whereby the message list can store memory offsets of message data stored in the message buffer.

In rejecting claim 8, the Examiner states that Erickson teaches the steps of creating a message list in the shared region for each process, whereby the message list can store memory offsets of message data stored in the message buffer (Col. 7, lines 1-55). Appellants respectfully disagree.

The RAM disclosed at col. 7, lines 1-55 is dual-port RAM 20, which the microprocessor 30 and CDCC 100 access from opposite sides. (Col. 7, lines 12-15). Thus, it appears that the Examiner equates Erickson's microprocessor 30 to the claimed first process and Erickson's CDCC 100 to the second process. Accordingly, to suggest the recited limitation, the microprocessor 30 and CDCC 100 each must have a message list created in the shared region of RAM. However, this is not the case.

Erickson does not suggest that the microprocessor 30 even has a message list. To the extent that the Examiner may assert that the Word-In-Queue register 112 is equivalent to the message list of the first process, it should be noted that the Word-In-Queue register 112 is part of the CDCC 100. Col. 7, line 46, *see also* FIG. 5A. Moreover, Erickson provides no teaching or suggestion that the Word-In-Queue register is created in the shared region of RAM. Indeed, as shown in FIG. 5A, the Word-In-Queue Register 112 is not contained in the RAM, but instead is an entirely different structure contained in the CDCC 100. Consequently, the Examiner has filed to establish a *prima facie* case of obviousness and claim 8 is distinguishable from the cited art. Accordingly, the rejection of claim 8 is improper and must be reversed. In addition, claim 8 is believed to be allowable by virtue of its dependence on allowable base claims.

H. <u>Claims 19 and 20</u>

1. <u>The cited references do not teach or suggest locking the accumulated data to prevent the first process from accessing the data while the accumulated data is being manipulated.</u>

In rejecting claims 19 and 20, the Examiner states that Bohannon, et al. teaches the step of locking the accumulated data to prevent the first process from accessing the data while the accumulated data is being manipulated (col. 1, lines 46-50). Appellant respectfully disagrees. The locking system in Bohannon, et al. provides a lock indicating exclusive access to the shared resources by a single process and enabling one or more processes to repeatedly attempt to gain access to the lock when ownership of the shared resource is desired. (Col. 1, lines 46-50). If the lock is busy, the process attempting to gain access to the lock can either relinquish its desire to obtain the lock so it can do other work, or it can wait or "spin" until the lock is released. (Col. 1, lines 51-

58). The method includes generating a linked list queue structure containing a first process currently having exclusive access to the lock and one or more processes added to the queue structure to spin on the lock, wherein each process is capable of modifying the queue structure by obtaining exclusive access of the lock from a process having exclusive access of the lock and releasing the lock to another process in the linked list queue structure, detecting when one or more processes having exclusive access to the lock terminates, and upon detection, removing the terminated process from the queue structure, and restoring consistency to said linked list queue structure. (col. 6, lines 49-67 and col. 7, lines 1-20).

In contrast, there is only the first process in Appellants' claimed invention prevented from accessing accumulated data in the location in the message buffer. (Appellants' specification, page 12, lines 19-24). There is no queue where the first process having exclusive access to the lock and one or more processes are added to the queue structure to spin on the lock. Furthermore, there is no requirement that the shared resource is specifically for accumulating message data or that the accumulated data is locked while the accumulated data is being manipulated as is required by claims 19 and 20. Consequently, the Examiner has filed to establish a *prima facie* case of obviousness and claims 19 and 20 are distinguishable from the cited art. In addition, claims 19 and 20 are believed to be allowable by virtue of their dependence on allowable base claims. Accordingly, the rejection of claims 19 and 20 is improper and must be reversed.

CONCLUSION

Appellants' invention relates to a method and apparatus for high speed interprocess communications ("IPC"). The high speed IPC method and apparatus of the present invention improves communication efficiency by passing data from a first process to a second process using a memory offset. The memory offset is used by the second process to access the data without requiring the data to be moved. Thus, the process eliminates the need for copying the data from the shared region of RAM to an alternate location while data manipulation takes place.

In the Examiner's rejection of claims 1-20, the cited references have been misguidedly applied. The cited references fail to render Appellants' invention obvious. Accordingly, Appellants submit that claims 1-20 define a patentably distinguishable invention over the prior art made of record, and a Notice of Allowance for claims 1-20 is accordingly and courteously solicited.

Respectfully submitted,

Date: 6-19-06

Robert J. Sacco
Registration No. 35,667
Sacco & Associates, P.A.
P.O. Box 30999
Palm Beach Gardens, FL 33420-0999
Tel: 561-626-2222

APPENDIX

CLAIMS

1. A method for high speed interprocess communications comprising the steps of:

attaching first and second processes to a message buffer in a shared region of random access memory (RAM) exclusive of operating system kernel space, each said process having a message list;

accumulating message data from said first process in a location in said message buffer;

said first process adding to said message list of said second process a memory offset corresponding to said location in said message buffer; and,

manipulating in said second process said accumulated data at said location corresponding to said offset,

whereby said accumulated message data is transferred from said first process to said second process with minimal data transfer overhead.

2. The method according to claim 1, wherein the attaching step comprises the steps of:

detecting a previously created shared region of RAM;

if a shared region of RAM is not detected, creating and configuring a shared region of RAM for storing accumulated data; and,

attaching said first and second processes to said shared region.

3. The method according to claim 1, wherein said message list is a message queue.

4. The method according to claim 3, wherein the adding step comprises the steps of:

retrieving a memory offset in said message buffer corresponding to said location of data accumulated by said first process; and,

inserting said memory offset in said message queue corresponding to said second process.

5. The method according to claim 4, wherein the inserting step comprises the step of atomically assigning said memory offset to an integer location in said message queue corresponding to said second process.

6. The method according to claim 1, wherein said manipulating step comprises the steps of:

identifying a memory offset in said message list corresponding to said second process;

processing in said second process message data stored at a location in said message buffer corresponding to said memory offset; and,

releasing said message buffer.

7. A method for configuring high speed interprocess communications between first and second processes comprising the steps of:

disposing a message buffer in a shared region of random access memory (RAM) shared between said first and second processes;

accumulating message data from said first process in a location in said message buffer;

said first process adding to a message list corresponding to said second process a memory offset corresponding to said location in said message buffer; and,

manipulating in said second process said accumulated message data stored in said message buffer at a location corresponding to said offset,

whereby said accumulated message data is transferred from said first process to said second process with minimal data transfer overhead.

8. The method according to claim 7, wherein the disposing step comprises the steps of:

creating and configuring a message buffer in a shared region of RAM exclusive of operating system kernel space; and,

creating a message list in said shared region for each said process,

whereby said message list can store memory offsets of message data stored in said message buffer.

9. The method according to claim 7, wherein said message list is a message queue.

10. The method according to claim 9, wherein the adding step comprises the steps of:

retrieving a memory offset in said message buffer, said memory offset corresponding to said location of said message data accumulated by said first process; and,

inserting said memory offset in said message queue corresponding to said second process.

11. The method according to claim 10, wherein the inserting step comprises the step of atomically assigning said memory offset to an integer location in said message queue corresponding to said second process.

12. The method according to claim 7, wherein said manipulating step comprises the steps of:

identifying a memory offset in said message list corresponding to said second process;

processing in said second process said accumulated message data at a location in said message buffer corresponding to said memory offset; and,

releasing said message buffer.

13. A computer apparatus programmed with a set of instructions stored in a fixed medium for high speed interprocess communications, said programmed computer apparatus comprising:

means for attaching first and second processes to a message buffer in a shared region of random access memory (RAM) exclusive of operating system kernel space, each said process having a message list;

means for accumulating message data from said first process in a location in said message buffer;

means for said first process to add to said message list of said second process a memory offset corresponding to said location in said message buffer; and,

means for manipulating in said second process said accumulated data at said location corresponding to said offset.

14. The computer apparatus of claim 13, wherein the attaching means comprises:

means for detecting a previously created shared region of RAM; and,

means for creating and configuring a shared region in RAM for storing accumulated data if a previously created shared region of RAM is not detected by said detecting means.

15. The computer apparatus according to claim 13, wherein said message list is a message queue.

16. The computer apparatus according to claim 15, wherein the adding means comprises:

means for retrieving a memory offset in said message buffer corresponding to said location of data accumulated by said first process; and,

means for inserting said memory offset in said message queue corresponding to said second process.

17. The computer apparatus according to claim 16, wherein the inserting means comprises means for atomically assigning said memory offset to an integer location in said message queue corresponding to said second process.

18. The computer apparatus according to claim 13, wherein said manipulating means comprises:

means for identifying a memory offset in said message list corresponding to said second process;

means for using in said second process message data at a location in said message buffer corresponding to said memory offset; and,

means for releasing said message buffer.

19. The method according to claim 1, further comprising the step of locking said accumulated data to prevent said first process from accessing said accumulated data while said accumulated data is being manipulated.

20. The method according to claim 13, wherein said accumulated data is locked to prevent said first process from accessing said accumulated data while said accumulated data is being manipulated.

EVIDENCE APPENDIX

(none)

RELATED PROCEEDINGS APPENDIX

(none)